Pointers and Structures

Kevin Miller

Pointers

Pointers are used to store the address of a variable.

Pointers are declared using the * symbol

Eg.

```
char c, *cp;
int i,    *ip;
float f, *fp;
double d, *dp;
```

c, i, f, d are normal variables each holding a different type of value
cp, ip, fp and dp are all pointers each holding an address

Pointers

```
char c, *cp;
int i,     *ip;
float f, *fp;
double d, *dp;
```

How big are these variables?

How many bytes does a pointer uses?

Pointers

Label	Address	Value
c	400	
cp	401–404	
i	405–408	
ip	409–412	
f	413–416	
fp	417–420	
d	421–428	
dp	429–432	

All addresses requires
4 bytes (32 bit system)

Therefore, all pointers, regardless of the type of variable pointed to, will require 4 bytes

The ampersand symbol (&) indicates the 'address of' a variable.

The asterisk (*) indicates 'at the address given by' a variable

Pointers

```
cp = &c;
ip = &i;
*ip = 42;
```

We begin by storing the address of variables c and i into pointer variables cp and ip

We then look at the address in ip which is 405 and place the value 42 at that address.

Label	Address	Value
c	400	
cp	401–404	400
i	405–408	42
ip	409–412	405
f	413–416	
fp	417–420	
d	421–428	
dp	429–432	

Pointers

We can also point to a cell in an array

```
char ca[3],*cp;
ca[1]=3;
cp=&(ca[1]);
*cp=7;
```

The memory map is given below:

Label	Address	Value
ca[0]	400	
ca[1]	401	~~3~~ 7
ca[2]	402	
cp	403–406	401

Pointer Arithmetic

When adding or subtracting amounts from an address, the arithmetic is done in terms of quantity of bytes that is equal to the size of the thing being referenced.

```
char ca[3],*cp;
int ia[3],*ip;
cp=&(ca[0]);
ip=&(ia[0]);
```

Label	Address	Value
ca[0]	400	
ca[1]	401	
ca[2]	402	
cp	403–406	400
ia[0]	407–410	
ia[1]	411–414	
ia[2]	415–418	
ip	419–422	407

Pointer Arithmetic

Let's consider the pointer arithmetic below:

```
*(cp+2)=8;   /* cp + 2 what? */
*(ip+2)=33;  /* ip + 2 what? */
```

Pointer Arithmetic

```
cp+2 = cp+2 (1 byte units) = 400+2 = 402
ip+2 = ip+2 (4 byte units) = 407+8 = 415
```

Label	Address	Value
ca[0]	400	
ca[1]	401	
ca[2]	402	8
cp	403–406	400
ia[0]	407–410	
ia[1]	411–414	
ia[2]	415–418	33
ip	419–422	407

Notice that using pointer arithmetic, the offsets match the indices of the arrays. This is the whole point.

Pointers and arrays can be used interchangeably; in fact, they are often the same thing.

Pointer to a pointer (double pointer)

In defining a pointer to a pointer, the first pointer contains the address of the second pointer, which points to the location that contains the actual value.

We declare a pointer to a pointer as follow:

<type> **<variable>

Eg:

```
int  ** var
```

which means `*(*var)`

Pointer to a pointer

Example:

```c
#include <stdio.h>

int main ()
{
   int  var;
   int  *ptr;
   int  **pptr;

   var = 3000;

   /* take the address of var */
   ptr = &var;

   /* take the address of ptr using address of operator & */
   pptr = &ptr;

   /* take the value using pptr */
   printf("Value of var = %d\n", var );
   printf("Value available at *ptr = %d\n", *ptr );
   printf("Value available at **pptr = %d\n", **pptr);

   return 0;
}
```

```
Value of var = 3000

Value available at *ptr = 3000

Value available at **pptr = 3000
```

Passing pointers to functions in C

```c
#include <stdio.h>
#include <time.h>

void getSeconds(unsigned long *par);

int main ()
{
   unsigned long sec;


   getSeconds( &sec );

   /* print the actual value */
   printf("Number of seconds: %ld\n", sec );

   return 0;
}

void getSeconds(unsigned long *par)
{
   /* get the current number of seconds */
   *par = time( NULL );
   return;
}
```

```
Number of seconds :1294450468
```

Notice how the value that is changed inside the function is reflected in the calling function.

Also note the use of the function declaration. This tells the compiler about a function name and how to call the function.

Return pointers from functions in C

To return an array from a function, you have to declare the return type as an array.

eg

```
int *
myFunction()
{
.
.
.
}
```

NB: It's a bad practice to return the address of a local variable outside of its function. Hence, you have to declare the local variable as a static variable.

Return pointers from functions in C

```c
#include <stdio.h>
#include <time.h>

/* function to generate and retrun random numbers. */
int * getRandom( )
{
   static int  r[10];
   int i;

   /* set the seed */
   srand( (unsigned)time( NULL ) );
   for ( i = 0; i < 10; ++i)
   {
      r[i] = rand();
      printf("%d\n", r[i] );
   }

   return r;
}

/* main function to call above defined function */
int main ()
{
   /* a pointer to an int */
   int *p;
   int i;

   p = getRandom();
   for ( i = 0; i < 10; i++ )
   {
       printf("*(p + [%d]) : %d\n", i, *(p + i) );
   }

   return 0;
}
```

Function to generate 10 random numbers and return them using an array name which represents a pointer ie address of first array element.

```
1523198053
1187214107
1108300978
430494959
1421301276
930971084
123250484
106932140
1604461820
149169022
*(p + [0]) : 1523198053
*(p + [1]) : 1187214107
*(p + [2]) : 1108300978
*(p + [3]) : 430494959
*(p + [4]) : 1421301276
*(p + [5]) : 930971084
*(p + [6]) : 123250484
*(p + [7]) : 106932140
*(p + [8]) : 1604461820
*(p + [9]) : 149169022
```

Structures in C

Since arrays are used to hold several data items of the same type, there is the need for a mechanism to hold data items of different types. This is where a Structure becomes useful. A structure defines a new data type.

```
struct [structure tag]
{
   member definition;
   member definition;
   ...
   member definition;
} [one or more structure variables];
```

- The structure tag is optional.
- The members are any valid variable definition.
- Structure variables are also optional.

Structures in C

Example of a book structure:

```
struct Books
{
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
} book;
```

The member access operator (.) is used to access the members of a structure.

Structures in C

```c
#include <stdio.h>
#include <string.h>

struct Books
{
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
};

int main( )
{
   struct Books Book1;        /* Declare Book1 of type Book */
   struct Books Book2;        /* Declare Book2 of type Book */

   /* book 1 specification */
   strcpy( Book1.title, "C Programming");
   strcpy( Book1.author, "Nuha Ali");
   strcpy( Book1.subject, "C Programming Tutorial");
   Book1.book_id = 6495407;

   /* book 2 specification */
   strcpy( Book2.title, "Telecom Billing");
   strcpy( Book2.author, "Zara Ali");
   strcpy( Book2.subject, "Telecom Billing Tutorial");
   Book2.book_id = 6495700;

   /* print Book1 info */
   printf( "Book 1 title : %s\n", Book1.title);
   printf( "Book 1 author : %s\n", Book1.author);
   printf( "Book 1 subject : %s\n", Book1.subject);
   printf( "Book 1 book_id : %d\n", Book1.book_id);

   /* print Book2 info */
   printf( "Book 2 title : %s\n", Book2.title);
   printf( "Book 2 author : %s\n", Book2.author);
   printf( "Book 2 subject : %s\n", Book2.subject);
   printf( "Book 2 book_id : %d\n", Book2.book_id);

   return 0;
}
```

```
Book 1 title : C Programming
Book 1 author : Nuha Ali
Book 1 subject : C Programming Tutorial
Book 1 book_id : 6495407
Book 2 title : Telecom Billing
Book 2 author : Zara Ali
Book 2 subject : Telecom Billing Tutorial
Book 2 book_id : 6495700
```

Structures as Function Arguments

```c
#include <stdio.h>
#include <string.h>

struct Books
{
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
};

/* function declaration */
void printBook( struct Books book );
int main( )
{
   struct Books Book1;/* Declare Book1 of type Book */
   struct Books Book2;/* Declare Book2 of type Book */

   /* book 1 specification */
   strcpy( Book1.title, "C Programming");
   strcpy( Book1.author, "Nuha Ali");
   strcpy( Book1.subject, "C Programming Tutorial");
   Book1.book_id = 6495407;

   /* book 2 specification */
   strcpy( Book2.title, "Telecom Billing");
   strcpy( Book2.author, "Zara Ali");
   strcpy( Book2.subject, "Telecom Billing Tutorial");
   Book2.book_id = 6495700;

   /* print Book1 info */
   printBook( Book1 );

   /* Print Book2 info */
   printBook( Book2 );

   return 0;
}
void printBook( struct Books book )
{
   printf( "Book title : %s\n", book.title);
   printf( "Book author : %s\n", book.author);
   printf( "Book subject : %s\n", book.subject);
   printf( "Book book_id : %d\n", book.book_id);
}
```

```
Book title : C Programming
Book author : Nuha Ali
Book subject : C Programming Tutorial
Book book_id : 6495407
Book title : Telecom Billing
Book author : Zara Ali
Book subject : Telecom Billing Tutorial
Book book_id : 6495700
```

Pointers to Structures

A pointer to a structure can be declared as:

```
struct Books *struct_pointer;
```

We can store the address of a structure in the pointer above like this:

```
struct_pointer = &Book1;
```

Accessing the members of a structure using a pointer to that structure, we use the -> operator as follows:

```
struct_pointer->title;
```

Pointers to Structures

Using the same book example above, but using structure pointer:

```c
#include <stdio.h>
#include <string.h>

struct Books
{
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
};

/* function declaration */
void printBook( struct Books *book );
int main( )
{
   struct Books Book1;/* Declare Book1 of type Book */
   struct Books Book2;/* Declare Book2 of type Book */

   /* book 1 specification */
   strcpy( Book1.title, "C Programming");
   strcpy( Book1.author, "Nuha Ali");
   strcpy( Book1.subject, "C Programming Tutorial");
   Book1.book_id = 6495407;

   /* book 2 specification */
   strcpy( Book2.title, "Telecom Billing");
   strcpy( Book2.author, "Zara Ali");
   strcpy( Book2.subject, "Telecom Billing Tutorial");
   Book2.book_id = 6495700;

   /* print Book1 info by passing address of Book1 */
   printBook( &Book1 );

   /* print Book2 info by passing address of Book2 */
   printBook( &Book2 );

   return 0;
}
void printBook( struct Books *book )
{
   printf( "Book title : %s\n", book->title);
   printf( "Book author : %s\n", book->author);
   printf( "Book subject : %s\n", book->subject);
   printf( "Book book_id : %d\n", book->book_id);
}
```

```
Book title : C Programming
Book author : Nuha Ali
Book subject : C Programming Tutorial
Book book_id : 6495407
Book title : Telecom Billing
Book author : Zara Ali
Book subject : Telecom Billing Tutorial
Book book_id : 6495700
```